Experiences with an Algorithm Recommender System

Naren Ramakrishnan
Department of Computer Science
Virginia Tech, VA 24061
Tel: (540) 231-8451
naren@cs.vt.edu

INTRODUCTION

There is now a significant amount of mathematical and scientific software available on the web. Much of it is accessible if we know 'what we want' and 'where it is'. However, the obstacles of selecting the best algorithm for a particular problem and subsequently finding an appropriate software implementation are often difficult and sometimes even impossible to surmount. As an example, it is estimated that there are nearly 10 million software modules for numerical quadrature that are potentially interesting and significantly different from one another! This paper outlines experiences with an algorithm recommender system — PYTHIA — that selects software modules on the web, given problem instances and user specified constraints.

One of the main research issues in algorithm recommendation is understanding the fundamental processes by which knowledge about scientific problems and their solutions is created, validated and communicated. While some of this knowledge could come from experts the field, other knowledge would be mined from experimental data or learnt by the system automatically as a result of experience.

ALGORITHM RECOMMENDATION

Algorithm recommendation is similar to traditional applications of recommender systems (like book and movie recommenders) in many ways. For instance, both of them embody the notion of profiling: a recommender system is only as effective and accurate as its user profiles, their coverage of the application domain and the performance of the recommender improves with 'experience'. However, algorithm recommendation also entails significant differences and interesting problems:

- While there is an inherent uncertainty in interpreting and assessing the performance measures of a particular algorithm for a particular problem, ratings and/or evaluations can be computed objectively and automatically by performance evaluation systems and it is not necessary to involve a human in the loop. For example, the performance database server [4] and the NetSolve project (at Oak Ridge National Labs) [1] together provide a facility to run experiments and catalog benchmark performance data on standardized test problems. Within each sub-domain, there is considerable agreement on what performance measures are deemed important, how test collections should be organized, what kinds of features (of problems) are considered relevant and so on.
- An algorithm recommender system is often required to justify the rationale for its recommendation (scientists and engineers would be experts at building models in their particular domain, but novices at understanding the intricacies of these mathematical models and the software systems required to solve them). It thus needs to communicate effectively with user(s) who will have very little background of actual algorithms and their performance. Furthermore, such a system is expected to provide 'What-If' design scenarios that explore the space of possible alternatives for recommendations.
- In traditional recommenders, features of the problem instance are easily obtained, either explicitly or via simple 'parsing' of the input. However, the cost and complexity to identify, characterize and measure features is very high in algorithm recommendation. For example, to test numerically if a matrix is 'positive definite' is usually much more expensive than the actual computation to be performed! Thus the usefulness of this feature is limited to those cases where 'positive definiteness' is measured from symbolic apriori information.
- Depending on the way the problem is (re)presented, the space of applicable algorithms changes; some of the best algorithms sacrifice generality for performance and have specially customized data structures and routines fine tuned for particular problems or their reformulations. A good algorithm recommender system should take this into account while assisting the user.

```
I        Differential and Integral Equations
I1         Ordinary Differential Equations (ODEs)
...
I2         Partial Differential Equations (PDEs)
I2a          Initial Boundary Value Problems
I2a1           Parabolic
...
I2b          Elliptic Boundary Value Problems
I2b1           Linear
I2b1a            Second Order
I2b1a1             Poisson (Laplace) or Helmholtz Equation
I2b1a1a              Rectangular Domain
...
I3         Integral Equations
```

Figure 1: Partial View of the GAMS Taxonomy.

DESIGN OF THE PYTHIA SYSTEM

The PYTHIA scientific recommender system [5] uses a database of problems (and their features), algorithms (and their characteristics) to mine rules that correlate the effect of problem features with algorithm performance. In addition, it interfaces with the GAMS mathematical software repository (http://gams.nist.gov) to provide an URL where a software module implementing this algorithm can be obtained. GAMS provides convenient access to thousands of software modules physically distributed among several Internet repositories. It supports search for appropriate modules by problem, keyword or name. Objects that can be downloaded include abstracts for the routines, documentation, examples and source code. GAMS's main contribution to scientific software is its tree structured taxonomy that helps determine appropriate modules for problem classes. A partial view of the taxonomy is shown in Fig. 1.

As shown, the problem class I refers to modules for solving differential and integral equations, I2 indexes to modules for PDEs, I2b is for elliptic boundary value problems, I2b1 is for linear elliptic boundary value problems and so on. GAMS functions in an interactive mode, guiding the user from the top of a classification tree to specific modules as the user describes the problem in increasing detail. During this process, many features of the problem are determined, indirectly from the user. However, at the 'leaves', there still exist several choices of algorithms for a specific problem instance. The entire GAMS tree has around 750 nodes, indexes over $10,000$ software modules, and is quite elaborate.

The combined PYTHIA-GAMS system can now be publicly accessed at http://www.cs.purdue.edu/research/cse/pythia or through the traditional GAMS interface with a specially designed proxy web server (For details on the implementation, we refer the interested reader to [6]). PYTHIA recommends algorithms for three domains of the GAMS taxonomy: H2a (one dimensional integration), I2b1a1a and I2b1a3 (elliptic partial differential equations). We now detail some of the research issues involved in building recommender systems for scientific domains:

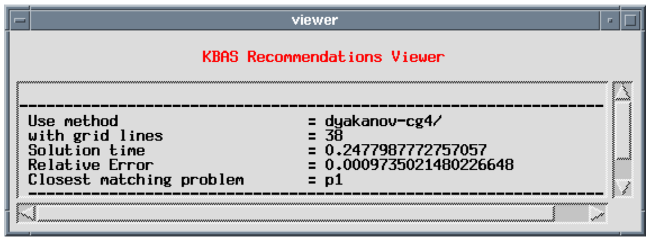
- AI techniques for organizing and selecting recommendations and the use of domain specific restrictions for the management of recommendation spaces. For example, PYTHIA uses syntactic and semantic restrictions on the nature of the induced hypotheses to reduce the search space of possible recommendations [8].
- Identifying various kinds of 'recommendation primitives' such as those involving link analysis, hybrid reasoning, constraint reasoning and constructive induction [5].
- Design of recommender systems around existing information infrastructures such as network repositories, cross indices and web-based information warehouses [6].
- Software kernels for creating recommender systems [2, 7].
- Information integration from multiple recommender systems [3]. PYTHIA's methodology is both content-based (in that recommendations are based on specific features of the user input) and collaborative (multiple recommender systems can interact and help to identify a good resource).
- Mining for recommendations from trace and performance data [8].
- Updating the basis for recommendations dynamically (allowing the system to function online) [8].

PYTHIA's design is intended to address many of these issues. In addition, the host problem-solving environment could monitor the progress of algorithms recommended and provide feedback data (about run-time exceptions, errors and unmet constraints) so that PYTHIA could update its 'basis' (the rule bank) automatically. A sample recommendation from PYTHIA might be: 'Use the 5-point star algorithm with a 200 X 200 grid on an NCube/2 with 16 processors: Confidence: 0.85, Software available at http://math.nist.gov/cgi-bin/gams-serve/list-module-components/ELLPACK/1-14-46/13058.html." For an empirical evaluation of PYTHIA, we refer the reader to [7].

INTERACTION WITH PYTHIA

PYTHIA is an important instructional aid for computational science. In particular, it uses terminological descriptions (also known as description logics) of domain entities to influence algorithm recommendation. This property is central for the creation of intelligent tutoring systems [9]. Such descriptions permit the use of subsumption and difference operators that are useful for providing explanations, decompositions and user modeling. Here are a few examples:

- Consider the concepts 'elliptic partial differential equations' (C1), 'partial differential equations' (C2) and 'elliptic' (C3). If PYTHIA needs to solicit informa-



KBAS Recommendations Viewer

```
---------------------------------------------------------------
Use method                          = dyakanov-cg4/
with grid lines                     = 38
Solution time                       = 0.2477987772757057
Relative Error                      = 0.0009735021480226648
Closest matching problem            = p1
---------------------------------------------------------------
```

Figure 2: Example recommendation from the PYTHIA system. In addition, PYTHIA identifies the closest matching problem in its database that has characteristics most similar to the one presented by the user.

tion pertaining to C1 to make a recommendation, and the user is an 'expert' in C2 (and C3), the difference operation $C1 - C2$ results in the description 'elliptic', thus decomposing the original problem into smaller subproblems that can be addressed by the user.

- Consider the case when a user identifies that algorithm X is good for any problem with 'a mixed boundary condition and a self-adjoint operator', whereas the recommender rules suggest that the algorithm is actually effective when the problem has just a 'mixed boundary condition'. PYTHIA can construct the difference that tells the user that 'self-adjoint operator' is not needed to determine the applicability of algorithm X.
- This facility can also be used to remove redundant information from descriptions (as shown in Fig. 2) where parts of information that the user already knows can be safely omitted (In the example here, the window does not show aspects of the computing environment, which were subsumed by the user's input description. The GAMS URL is also not specified since the user accessed the system through the GAMS web page.)
- This can also aid in collaboratively identifying a community of users with similar descriptions (like 'scientists who are familiar with adaptive algorithms but not non-adaptive algorithms'). In addition, PYTHIA can be used to create versions of GAMS tailored to specific applications.

FUTURE DIRECTIONS

We plan to expand PYTHIA into a tool for the entire national science and engineering community and to explore ways to extend our implementations to less structured domains of scientific software. PYTHIA would help scientists and engineers achieve increased levels of interactivity as they work together to solve common problems. Further, it will enable and hence encourage an increased flow of information and knowledge among these scientists, their organizations and professional communities.

ACKNOWLEDGMENTS

The author wishes to acknowledge helpful discussions with John M. Carroll, Virginia Tech and John R. Rice, Purdue University.

REFERENCES

1. Casanova, H. and Dongarra, J.J. NetSolve: A Network Server for Solving Computational Science Problems, Technical Report CS-95-313, University of Tennessee, 1995. Accessible at http://www.cs.utk.edu/netsolve.

2. Houstis, E.N., Verykios, V.S., Catlin A.C., Ramakrishnan, N. and Rice, J.R. PYTHIA II: A K/DB System for Recommending and Testing Scientific Software, ACM Transactions on Mathematical Software, Communicated. Also available as Technical Report CSD-TR-98-031, Department of Computer Sciences, Purdue University, 1998.

3. Joshi, A., Ramakrishnan, N. and Houstis, E.N., Multi-Agent System Support for Networked Scientific Computing, IEEE Internet Computing, Vol. 2(3), pages 69-83, 1998.

4. LaRose, B. The Design and Implementation of a Performance Database Server, Technical Report CS-93-195, University of Tennessee, 1993. Accessible at http://performance.netlib.org/performance/html/PDStop.html.

5. Ramakrishnan, N. Recommender Systems for Problem Solving Environments, Ph.D. Thesis, Department of Computer Sciences, Purdue University, August 1997.

6. Ramakrishnan, N., Houstis, E.N., Joshi, A., Rice, J.R. and Weerawarana, S. Intelligent Networked Scientific Computing, In Proceedings of the 15th IMACS World Congress, pages 785-790, Wissensshaft and Technik Verlag, 1997.

7. Ramakrishnan, N., Houstis, E.N. and Rice, J.R. Recommender Systems for Problem Solving Environments, In H. Kautz, Editor. Working Notes of the AAAI-98 Workshop on Recommender Systems, AAAI/MIT Press, 1998.

8. Ramakrishnan, N., Rice, J.R. and Houstis, E.N. GAUSS: An Online Algorithm Recommender System for One-Dimensional Numerical Quadrature, ACM Transactions on Mathematical Software, Communicated. Also available as as Technical Report CSD-TR-96-031, Department of Computer Sciences, Purdue University, 1996.

9. Teege, G. Making the Difference: A Subtraction Operator for Description Logics. In J. Doyle, E. Sandewall and P. Torasso, Editors. Principles of Knowledge Representation and Reasoning: Proc. of the 4th International Conference (KR 94), Morgan Kaufmann, San Francisco, CA, 1994.